Proceedings of the Korean Nuclear Society Spring Meeting
Pohang, Korea, May 1999

Software Error Masking Effect on Hardware Faults

Jong Gyun Choi and Poong Hyun Seong

Korea Advanced Institute of Science and Technology
Department of Nuclear Engineering
373-1 Kusong-dong, Yusong-gu
Taejon, Korea 305-701

Abstract

Based on the Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL), in this work, a simulation model for fault injection is developed to estimate the dependability of the digital system in operational phase. We investigated the software masking effect on hardware faults through the single bit-flip and stuck-at-x fault injection into the internal registers of the processor and memory cells. The fault location reaches all registers and memory cells. Fault distribution over locations is randomly chosen based on a uniform probability distribution. Using this model, we have predicted the reliability and masking effect of an application software in a digital system-Interposing Logic System (ILS) in a nuclear power plant. We have considered four the software operational profiles. From the results it was found that the software masking effect on hardware faults should be properly considered for predicting the system dependability accurately in operation phase. It is because the masking effect was formed to have different values according to the operational profile.

1. Introduction

Computers today form integral parts of large systems where processing and control are the primary demands. They can also be the mainstay of systems, such as flight control systems and nuclear protection systems, which are required to be ultra-reliable. Designing the reliable system and predicting the system reliability accurately are, therefore, the most important issues that the computer designers and developers face.

The dependability estimation of the digital systems by fault injection has become key issues and has drawn the attention of many researchers because the complexity of digital system including fault tolerant mechanisms make analytical estimation of system very difficult. Fault injection is an important technique for the evaluation of design metrics such as reliability, safety, and fault coverage. Fault injection involves inserting faults into a system and monitoring the system to determine its behavior in response to the fault. Several approaches for fault injection have been made and fall into three categories: fault injection at the physical level, software implemented fault injection and simulation-based fault injection. Injecting fault at the physical level has been accomplished by inducing soft errors with heavy-ion radiation to several processors,[1] inducing electro-magnetic interference to the hardware and corrupting the values of an IC pins during several clock cycles in order to ensure that fault would manifest itself as an error for at least one cycle.[2] One of software implemented fault injection techniques injects transient faults by corrupting the process' s memory image and by inserting software trap instructions.[3] The simulation-based approaches are done at a gate-level[4,5] where signal values in the simulation are stuck at logic ' 1' or ' 0' or at a device-level[6] where current or voltage values are fixed.

Most of these researches however have focused on fault coverage and error latency of fault-tolerant mechanisms in computing systems as dependability measures. At recent years, it was reported that the hardware transient faults could be masked by only software without hardware error masking mechanisms. That is, a substantial number of faults do not affect the program results for several reasons: faults whose errors are neutralized by the next instructions, faults affecting the execution of instructions that do not contribute to the benchmark results, and faults whose errors are tolerated by the semantic of the benchmark under execution. This effect should be considered properly because even a small change of system fault coverage value can affect the system dependability. [7]

In this work, VHDL-based simulation model for fault injection is developed to estimate the reliability of the digital system in operational phase considering the software masking effect on hardware faults. The faults occurred in system must escape not only the fault tolerance mechanisms but also software masking of hardware faults to cause the failure of the system. Therefore, for estimating the dependability of digital system accurately in operational phase, operational input profile of software is considered because the software execution by different operational profile can mask the different type of faults.

We investigated the single bit-flip and stuck-at-x fault in the internal registers of the processor and in memory cells. The fault location covers all registers and memory cells. Fault distribution over locations is randomly chosen based on an uniform probability distribution. This model can estimate the reliability of the system more accurately in operational phase.

Using this model, we predict the reliability and the software coverage function of the application software in a digital system, Interposing Logic System (ILS)[8] in a nuclear power plant, whose diagram is shown in Figure 1. The system is a complete, self-contained one board computer system consisting of 8 bit, n-channel 8085 microprocessor, 8155 multipurpose peripheral chip, random access memory (RAM), read only memory (ROM), programmable timer, I/O ports, bus control logic, and memory expansion buffers.

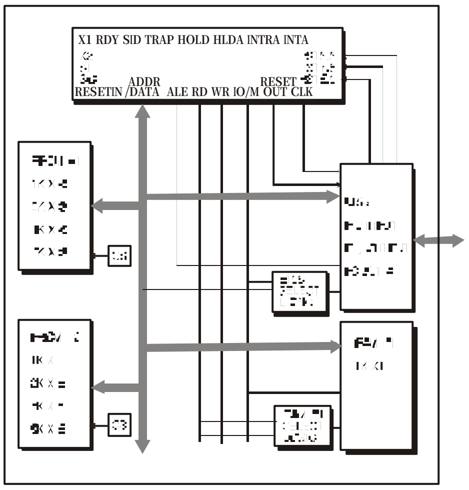
X1 RDY SID TRAP HOLD HLDA INTRA INTA
ADDR
RESETIN /DATA ALE RD WR IO/M
RESET
OUT CLK

Figure 1. Block diagram of a processor module

2. Model

2.1 Configuration of VHDL Simulation Model

As shown in Figure 2, a module (*RANDOM FAULT LOCATION AND TIME GENERATOR; RFLATG*) determines the location in registers of microprocessor and memory cells into which fault is injected. This module is coded by C language and produces the text file which contains the fault location and time. The fault injector module reads the data from the RFLATG module and determines the fault type (transient or permanent) of the corrupted location in components. Then, it injects the fault into the selected registers and memory cells. The fault injection technique in this module uses the bus resolution function provided by VHDL language. The *FAULT SELECTION LOGIC* determines the component into which the fault is injected. The software embedded in ROM reads the input data from the random access memory (RAM) and store the results into the RAM. Therefore, the operational profile of the software is determined by the contents in the region of RAM from which microprocessor fetch the data to execute the software instructions. The ***P**ROFILER* module changes the values in the input data region of RAM memory to simulate the system operation in operational phase. All modules are modeled at behavioral level with Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL).

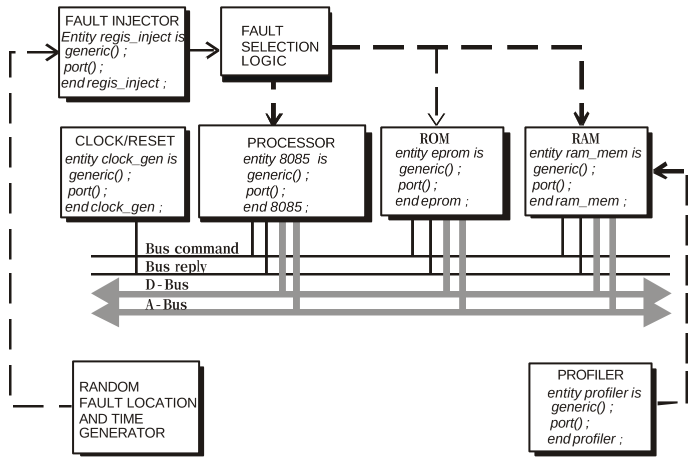

Figure 2. Block diagram of VHDL simulation model

2.2 Fault Injection Methodology

For injecting the stuck-at-x fault into the register cells of microprocessor and memory cells, a bus resolution function provided by VHDL language is used to resolve the value of the faulty signal with the normal signal of system. A resolution function is a function that defines how the values of multiple sources of a given signal are to be resolved into a single value for that signal.[9] Therefore, for the stuck-at-one fault injection, the bit value ‘1’ from the fault injector module is wired-or with the component in which the value should be corrupted. For the stuck-at-zero fault injection, the bit value ‘0’ from the fault injector module is wired-and with the component. The fault injection technique using the resolution function was applied by DeLong.[10] For the bit-flip fault, we followed the methodology proposed by Ward.[11] Each VHDL model, which describes normal behavior of components, contains sub-process associated with the faulty behavior when the bit flip fault occurs in component. Fault distribution over locations is randomly chosen with an uniform probability distribution.

2.3 Discrete Reliability

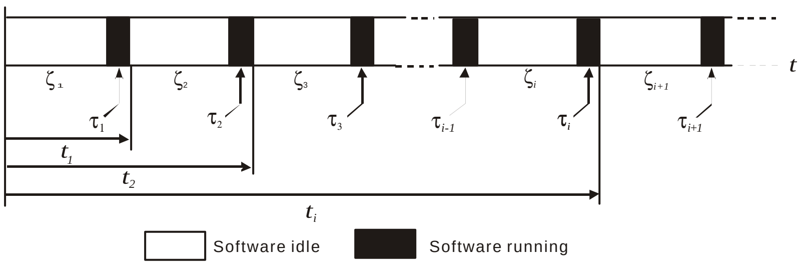

Figure 3. Relationship between the various time interval

As shown in Figure 3, the software embedded in computer system is either in idle state or in execution state. The software is idle during the time z_i between the *i-1*th and *i*th execution state. The *i*th execution of the software begins after this idle time z_i and continues for the execution time t_i. Since t_i is very small compared with the idle time z_i, it is reasonable to assume that t_1 » ... » t_i » ... » t. The software completes its first execution at time t_1 and the second execution of the software is finished at time t_2.

A. The fault coverage function

When n faults are injected into system, the fault coverage of hardware fault detection and recovery mechanism is modeled as a two-event discrete random variable $y_i(t)(i = 1, 2, .., n)$ defined as follows[12]:

$$y_i(t) = \begin{cases} 1, \text{ if ith fault is covered in } [0, t\,] \\ 0, \text{ otherwise} \end{cases} \tag{1}$$

The number of faults covered within the time interval [0,t] is as follows:

$$N_H(t) = \sum_{i=1}^{n} y_i(t) \tag{2}$$

and the hardware coverage function $C_H(t)$ can be estimated as

$$\hat{C}_H(t) = \frac{N(t)}{n} \tag{3}$$

In this case, the error state is signaled and faults are removed from the system. In addition, hardware faults could be removed or masked by only software without hardware error masking mechanisms. Therefore, to determine the system fault coverage correctly, the software error masking effect is considered. To include the software masking effect and estimate the reliability, the equations (1,2,3) is extended as follows:

$$y_i(t_x,t_y)=\begin{cases}1, & \text{if ith fault is covered in } [\mathrm{t_x}, \mathrm{t_y}] \\ & \quad \text{by hardware or software masking effect} \\ 0, & \text{otherwise}\end{cases} \tag{4}$$

$$N(t_x,t_y)=\sum_{i=1}^{n} y_i(t_x,t_y) \tag{5}$$

$$\hat{C}(t_x,t_y)=\frac{N(t_x,t_y)}{n} \tag{6}$$

When the system failure is defined as unacceptable output of the software, Equation (3) can be rewritten as discrete function. In Figure 3, if the fault is injected before time t_1, the coverage function at time t_1 is given by

$$\hat{C}_1(t_1)=\frac{\sum_{i=1}^{n} y_i(t_f,t_1)}{n} \equiv C_{1,1} \equiv C_1, \tag{7}$$

where t_f is fault injection time. Although faults injected before time t_1 are not removed by hardware fault tolerant mechanism, the acceptable results by software error masking can be delivered at time t_1. These faults are not removed in case of permanent fault type and become sources of the system failure during the second software execution. For faults injected before time t_1, The coverage function at time t_2 is given by

$$\hat{C}_1(t_2)=\frac{\sum_{i=1}^{n} y_i(t_f,t_2)}{n} \equiv C_{1,2} \equiv C_2, \tag{8}$$

Generally, the coverage function at time t_k is as follow:

$$\hat{C}_1(t_k)=\frac{\sum_{i=1}^{n} y_i(t_f,t_k)}{n} \equiv C_{1,k} \equiv C_k \tag{9}$$

If the faults are injected in [t_{k-1}, t_k], the coverage function at time t_k is equal to the equation (7) statistically.

$$\hat{C}_k(t_k)=\frac{\sum_{i=1}^{n} y_i(t_f,t_k)}{n} = C_{k,k} \cong C_{1,1} = C_1, \quad for \quad t_{k-1} < t_f \le t_k \tag{10}$$

$$\hat{C}_k(t_{k+1}) = \frac{\sum_{i=1}^{n} y_i(t_f, t_{k+1})}{n} \equiv C_{k,k+1} \cong C_{1,2} = C_2, \quad for \quad t_{k-1} < t_f \le t_k \tag{11}$$

Generally, the coverage function is as follows:

$$\hat{C}_k(t_{k+i}) = \frac{\sum_{i=1}^{n} y_i(t_f, t_{k+i})}{n} \equiv C_{k,k+i} \cong C_{1,i+1} \equiv C_{i+1}, \quad for \quad t_{k-1} < t_f \le t_k \tag{12}$$

B. Discrete Reliability Estimation

The reliability expression for a non-maintained fault-tolerant system can be written as:[12]

$$R(t) = 1 - \Phi_F(t) + \int_0^t f_F(t_F) C(t - t_F) dt_F, \tag{13}$$

where $\Phi_F(t)$ and $f_F(t_F)$ are the cumulative distribution and density function of the fault occurrence process of the target system respectively and $C(t)$ is probability density function of coverage function. This equation can be also rewritten as a discrete function. The reliability at time t_0 $(= 0)$ is given by

$$R(t_0) = 1 \tag{14}$$

and the reliability at time t_1, t_2 is given by

$$R(t_1) = 1 - \Phi_F(t_1) + [\Phi_F(t_1) - \Phi_F(t_0)] \cdot C_{1,1} = 1 - \Phi_F(t_1) + \Phi_{1,0} \cdot C_1, \tag{15}$$

$$\begin{aligned} R(t_2) &= 1 - \Phi_F(t_2) + [\Phi_F(t_1) - \Phi_F(t_0)] \cdot C_{1,2} + [\Phi_F(t_2) - \Phi_F(t_1)] \cdot C_{1,1} \\ &= 1 - \Phi_F(t_2) + \Phi_{1,0} \cdot C_2 + \Phi_{2,1} \cdot C_1 \end{aligned} \tag{16}$$

Generally, we have reliability at time t_i as the follows:

$$R(t_i) = 1 - \Phi_F(t_i) + \sum_{j=1}^{i} \left(\Phi_{j,j-1} \cdot C_{i+1-j} \right) \tag{17}$$

$$MTTF = \sum_{i=1}^{\infty} R(t_i)(t_i - t_{i-1}) \tag{18}$$

3. Model Application

The target ILS software is a part of AFS-1000 system developed by Forney International Cooperation and installed in YGN nuclear power units 3 and 4 in Korea. It is constructed in the Intel 8085 assembly language using top-down modular design techniques. Intel 8085 processor has the 15 registers (AC, P, S, CY, Z, ACC, B, C, D, E, H, L, IR, SP, PC).[13] On board memory capabilities include the two 1K x 1 read/write memories for single bit data storage and a 1K x 8 read/write memory for 8 bit data storage. Read-only-memory capability ranges from 1K bytes to the maximum 48K bytes. The software is used in a control system and is of very simple logic algorithm, which leads to the result, 'yes' or 'no'. In addition, the inputs of the software have only two kinds of values, 0 or 1. The 10,000 faults for each type of faults are injected into VHDL simulation model and ten cycles of software execution for each faults is inspected to determine whether the system fails or not.

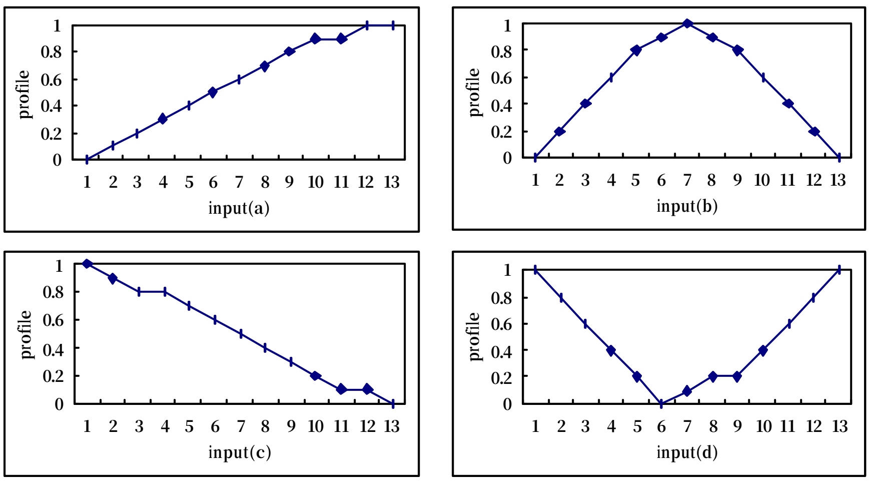

Figure 4. Operational profile of ILS

Figure 4 shows four cases of the software operational profile. The operational profiles of the software are determined by thirteen inputs. These four cases were prepared to show how the software masking effect changes for different operational profile. The y-axis is the probability that each inputs of the software has the value, 0. Figure 5 shows the software masking effect of stuck-at-x fault in registers to the four operational profile cases. For the stuck-at-zero fault, the system fails as soon as the fault occurs in registers of processor. 99.98 % of 10,000 faults cause the system failure for the first software execution. But, for

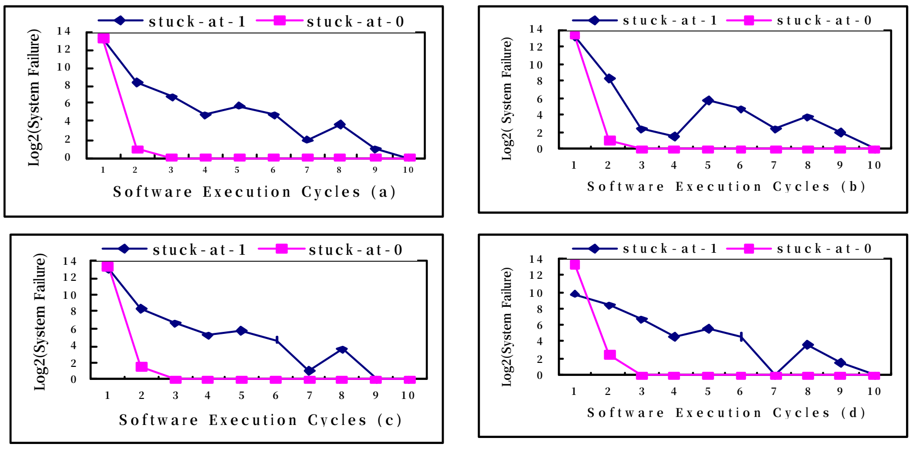

Figure 5. Software masking effect of stuck-at-X fault in registers

the stuck-at-1 faults, although 94.5 % of 10,000 faults induce the system failure during the first cycle, the software masks some faults and puts off the system failure for the a few software execution cycles. Specially, when the software runs by operation profile (d), the software masks the faults for period longer than ten cycles. Figure 6 shows the software masking effect of stuck-at-x fault in memory. In this experiment, the faults are injected into only region occupied by the software. The shape of curves is similar to that of stuck-at-zero fault in registers of microprocessor. Figure 7 shows the case of transient bit-flip fault injection into the registers of microprocessor. The system failure occurs only at the

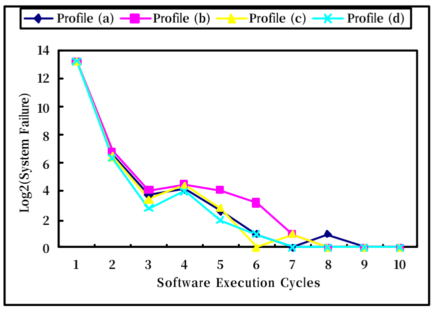

Figure 6. Software masking effect of stuck-at-X fault in memory

first cycle. In case of bit-flip fault injection experiment, it was shown that if the faults do not cause the system failure at the first cycle they are recovered by software. Table 1 shows the coverage value of stuck-at-X faults in memory by software error masking. Figure 8 shows the system reliability by memory faults with considering software error masking effect in case of operational profile (a). The memory fault rate is one per 10000 hr.

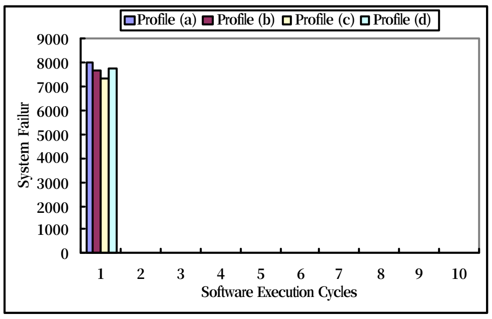

Figure 7. Software masking effect of bit-flip faults in registers

Coverage / Profile	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}
a	0.0138	0.0041	0.0028	0.001	0.0004	0.0002	0.0002	0	0	0
b	0.0186	0.0069	0.0052	0.0029	0.0012	0.0003	0.0001	0	0	0
c	0.0131	0.0043	0.0032	0.0011	0.0004	0.0003	0.0001	0	0	0
d	0.0113	0.0031	0.0024	0.0008	0.0004	0.0002	0.0001	0	0	0

Table 1. Software coverage of stock-at-X in memory

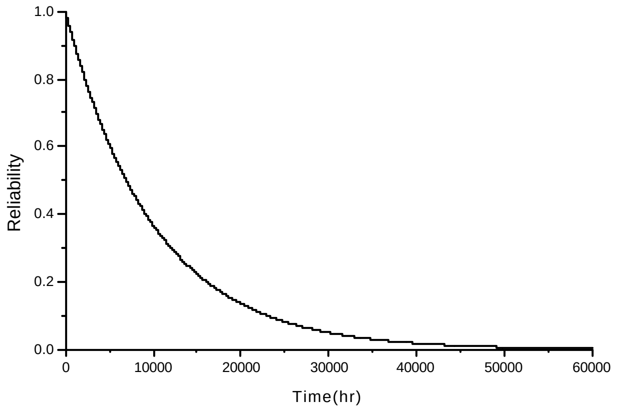

Figure 8. System Reliability by memory stuck-at-X fault

4. Summary and Conclusions

In case of this application software, some of stuck-at-one faults in registers can be masked during a few software execution cycles but the stuck-at-zero faults have very short latency time. That is, The software is sensitive to the stuck-at-zero fault in registers. The software has large coverage value for bit-flip faults in registers compared with other faults and removes the transient faults.

In this work, VHDL-based simulation model for fault injection is developed to estimate the reliability of the digital system in operational phase with considering the software masking effect of hardware faults. The faults occurred in system must escape not only the fault tolerance mechanisms but also software masking of hardware faults to cause the failure of the system. Therefore, for estimating the reliability of digital system accurately in operational phase, operational input profile of software is considered because the software execution by different operational profile can mask the different type of faults.

References

1. U. Gunneflo, J. Karlsson, and J. Torin : "Evaluation of error detection schemes using fault injection by heavy-ion radiation," Proc. 19th Symp. on Fault-Tolerant Computing (FTCS-19), Chicago, Illinois, 21-23, June 1989, pp. 340-347.
2. J. Karlsson, P. Folkesson, J. Arlat, Y. Crouzet, G. Leber, J. Reisinger : "Application of Three Physical Fault Injection Techniques to the Experimental Assessment of the MARS Architecture", Preprints of Fifth International Working Conference on Dependable Computing for Critical Applications (DCCA-5), Urbana-Champaign, Illinois, USA, Septemper 27-29, 1995, pp. 150-161.
3. J. Barton, E. Czeck, Z. Segall, D. Siewiorek, "Fault Injection Experiments using FIAT," IEEE Transaction on Computers, Vol. 39, No. 4, April 1990, pp. 572-582.
4. E. Czeck, D. Siewiorek, "Effects of Transient Gate-Level Faults on Program Behavior," Proc. 20th Symp. On Fault Tolerant Computing (FTCS-20), Newcastle upon Tyne, June 1990, pp. 236-243.
5. M. Rimen, J. Ohlsson, J. Torin, "On Microprocessor Error Behavior Modeling," Proc. IEEE 24th International Symposium on Fault-Tolerant Computing(FTCS-24), Austin, Texas, USA, 1994, pp.76-85.
6. Choi, S. Gwan, Ravishankar K. Iyer, "Focus : An Experimental Enviromnent for Fault Sensitivity analysis," IEEE Transaction on Computers, Vol. 41, No. 12, Dec. 1992, pp. 1515-1526.
7. Joanne B. Dugan, Kishor S. Trivedi, "Coverage Modeling for Dependability Analysis of Fault-Tolerant Systems," IEEE Transactions on Computers. Vol. 38, No. 6 June 1989.
8. KOPEC, "ILS System Technical Manual," 1994.
9. IEEE, "IEEE Standard VHDL Language Reference Manual," New York, NY, IEEE Std. 1076-1993, June 6, 1994.

10. T. DeLong, B. Johnson, J. Profeta, “A Fault Injection Technique for VHDL Behavioral-Level Models,” IEEE Design & Test of Computers, Vol. 13, No. 4, Winter 1996.
11. P. Ward, J. Amstrong, “Behavioral Fault Simulation in VHDL”, 27th ACM/IEEE Design Automation Conference – Proceedings 1990, pp. 587-593.
12. J. Arlat, A. Costes, Y. Crouzet, J.C. Laprie, D. Powell, “Fault Injection and Dependability Evaluation of Fault-Tolerant Systems, IEEE Transactions on Computers, 42 (8), pp.913-23, August 1993.
13. L. Leventhal, “8080A/8085 Assembly Language Programming,” Osborne/McGraw-Hill, Berkeley, California, 1978.